EECS 373 *Midterm 2*

Fall 2023

Name: _____ unique name: _____

Sign the honor pledge code:
"I have neither given nor received unauthorized aid on this examination, nor have I concealed any violations of the Honor Code"

Signature: _____

>> <u>Do not</u> open the exam until instructed to begin the exam <<

Exam Guidelines:

1. Closed book/notes. Scientific calculators are allowed. No graphing calculators or other electronic devices. Phones must stay in your pockets or bags.
2. There are 15 pages to this exam, including this one
3. You will be provided with the *Fall 2023 Exam 2 Datasheet*
4. You will be provided with the Reference Sheet aka "Appendix 1 & 2"
5. You have 120 minutes for this exam
6. Don't spend too much time on any one problem
7. Show your work and explain what you've done when asked to do so
8. State any assumptions
9. Getting partial credit without showing work will be rare

Question 1: Multiple Choice	20
Question 2: Short Answer	18
Question 3: Serial	10
Question 4: Interrupts	10
Question 5: Design Problem	-
Part 1: GPIO Initialization	4
Part 2: Joystick	8
Part 3: Timer Initialization	8
Part 4: Servo Position	8
Part 5: ADC Function	10
Part 6: Write the Control Loop	4
Total	**100**

Question 1: Multiple Choice [2 points each]

For each of the multiple choice questions select the most correct answer by fill in the square

Q 1.1

Which statement most accurately describes the difference between the ARMv7-M Architecture and STM32L4 ARM Cortex-M4.

- ☐ The ARMv7-M is an integrated circuit designed for the STM32L4
- ☐ The STM32L4 is a chip that has a physical ARM Cortex-M4 module on it
- ☑ The STM32L4 is an integrated circuit that implements the ARMv7-M architecture
- ☐ The STM32L4 and ARMv7-M are both chips on the Nucleo Dev Board that work together to form an embedded system
- ☐ The ARMv7-M and the STM32L4 are unrelated

Q1.2

Which is an example of a Memory Mapped I/O register?

- ☐ Direct Memory Access Controller
- ☐ Program Counter Register
- ☐ Link Register
- ☐ Flash and SRAM Memory
- ☑ SPI Receive Register

Q 1.3

When must data be stable for a correct I²C bus transaction?

- ☐ The rising edge of the clock
- ☐ The falling edge of the clock
- ☐ When the clock is in a high-impedance state
- ☐ When the clock is low
- ☑ When the clock is high

Q 1.4

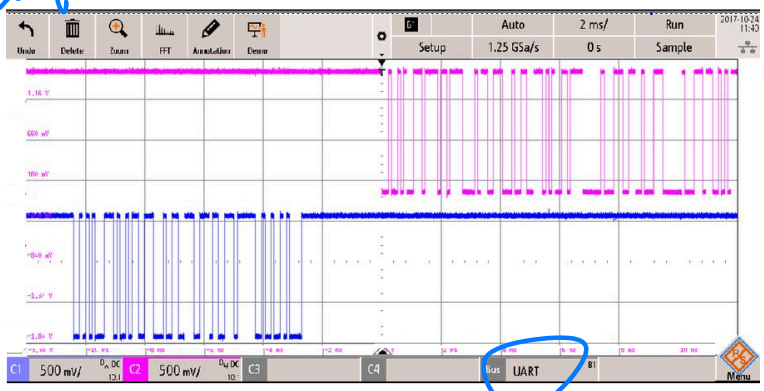
Given an 80MHz clock and 16-bit timer, what is the smallest prescaler possible that allows accurately measuring a pulse width between 3ms and 20ms?

- ○ 3
- ○ 13
- ○ 23
- ○ ~~24~~
- ○ 25

Q1.5

Which serial protocol is depicted in the following oscilloscope screenshot?

- [x] UART
- [ ] SPI
- [ ] I2C
- [ ] USB
- [ ] Can not be determined



Q1.6

Why is an output buffer (aka voltage follower opamp) needed on Digital-to-Analog Converters?

- [ ] Stabilize the comparator's threshold voltage during the conversion process
- [x] Keep the output of the DAC from being affected by the load
- [ ] Hold the input analog signal during the conversion process
- [ ] Sample that DAC staircase during the conversion process
- [ ] Sample and hold the output of the binary counter during the conversion process

Q1.7

Which statement is **NOT** true about the Interrupt Vector Table?

- [ ] The Interrupt Vector Table can be edited
- [x] It is stored in the Nested Interrupt Vector Controller (NVIC)
- [ ] It stores the locations of interrupt subroutines
- [ ] The first 16 values are defined by the ARM Company
- [ ] Memory location 0x0000 0004 holds the initial PC value

Q1.8

Which ADC can programmatically vary the number of bits used to represent an analog value?

- [ ] Flash
- [ ] PWM + LPF
- [ ] Slope Integrator
- [x] SAR
- [ ] Slope Integrator and SAR
- [ ] PWM + LPF, Slope Integrator, and SAR

Q1.9

An generic ARM CPU with the Interrupt Priority Register (IPR) with preemption priority and subpriority number is configured as shown below.

Interrupt Priority Register (IPR)

7	6	5	4	3	2	1	0
PRE			SUB			X	
r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w

Bits [7:5] **PRE:** preemption priority

Bits [4:2] **SUB:** subpriority

Bits [1:0] **X:** Do not care

This CPU is already handling an interrupt (IR0) with IPR_0 = 0xC4 via its interrupt service routine. While handling this interrupt, two more interrupts (IR1, IR2) arrive at the same time with IPR_1 = 0xE8 and IPR_2 = 0xE6. Which of the following is true?

- [ ] IR1 and IR2 will preempt IR0
- [ ] IR1 will preempt IR0 and IR2 will tail chain afterward
- [ ] This is not an example of preemption, tail chaining, or late arrivals
- [ ] IR1 is handled before IR2,
- [ ] IR0 service routine runs to completion, then IR2 is handled before IR1
- [ ] IR0 service routine runs to completion, but IR1 is dropped because IR2 is a late arrivals

Q1.10

What factors limit the maximum sampling frequency of the ADC on the STM32L4R5?

- [ ] Sample and hold time
- [ ] Conversion time
- [ ] Clock Speed
- [ ] APB bus speed
- [ ] All the above contribute
- [ ] None of the above

Question 2: Short Answers [3 points each]

Q2.1

Check 3 quantization error 不一定.

Consider an 8-bit ADC, where V_{REF-} = 0V and V_{REF+} = 5V. What is the minimum quantization error of this ADC? You may give your answers in the form of fractions.

$\frac{5V-0V}{2^8} = \frac{5V}{256} = 0.0195V$

$0.0195V/2 = 9.7 \times 10^{-3} V$.

Q2.2

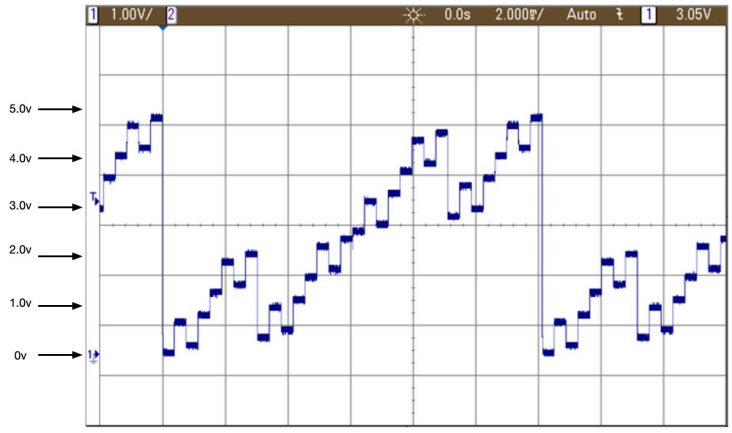
Explain the difference between preemption-priority and sub-priority of peripheral interrupts.

Preemption-priority will allow higher priority interrupt to preempt lower priority interrupt and have nested interrupts.

But sub-priority can't be used as the standard to do preemption. Instead it can only decide if several interrupts and they have the same preemption-priority, which one run first.

Q2.3

The following oscilloscope plot shows the output of a DAC with a Vref+ = 5v and Vref- = 0v. How many bits is the DAC? [2 points]



execution.

to allow programmers the flexibility to control interrupt ① The priority of the other interrupts are adjustable

Q2.4
The ARMv7-M Architecture requires that the Reset, NMI, and HardFault interrupts have a fixed priority of -3, -2, and -1 (respectively). Why are these interrupt priority levels negative and fixed while all other interrupts are positive and adjustable?

Since those interrupts are given by the ARM, they are defaun and build in interrupt. the user customized interrupt will start from 0. ① No programmer-defined interrupt can have higher importance than these interrupts. ② And they can not be changed.

Q2.5
You have an up-counting timer with a 1MHz reference clock. You set the prescaler value to 3. Find the values of the Auto-Reload Register (ARR) and Capture/Compare Register (CCR) to create a 50 kHz, 20% duty cycle PWM signal.

$\frac{1}{1\times10^6} \times 4 \times (ARR+1) = \frac{1}{50\times10^3}\,s$

$\frac{1}{1\times10^6} \times 4 \times ARR = \frac{1}{50\times10^3}$.

ARR = 4

CCR = 1

50 kHz

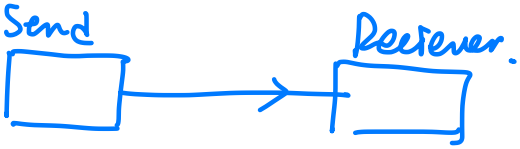
$\frac{1}{50\times10^3\,s}$

$\frac{1}{50000} \times ARR = \frac{1}{5\times10^4}$

ARR = 50000 × 1/5 = 10000

Q2.6
You are asked to diagnose a misbehaving UART module on a microcontroller that is sending garbled and incorrect characters to a computer terminal. The receiver hardware for the computer works correctly and you are 100% certain that the Baud Rate is set correctly in code for both the sender and receiver. State one reason why the computer is receiving garbled and incorrect characters? [3 points]

去总结 UART. SPI. I2C 的区别.

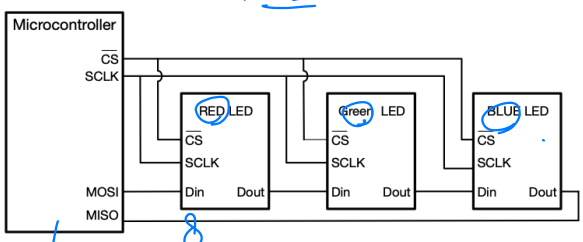


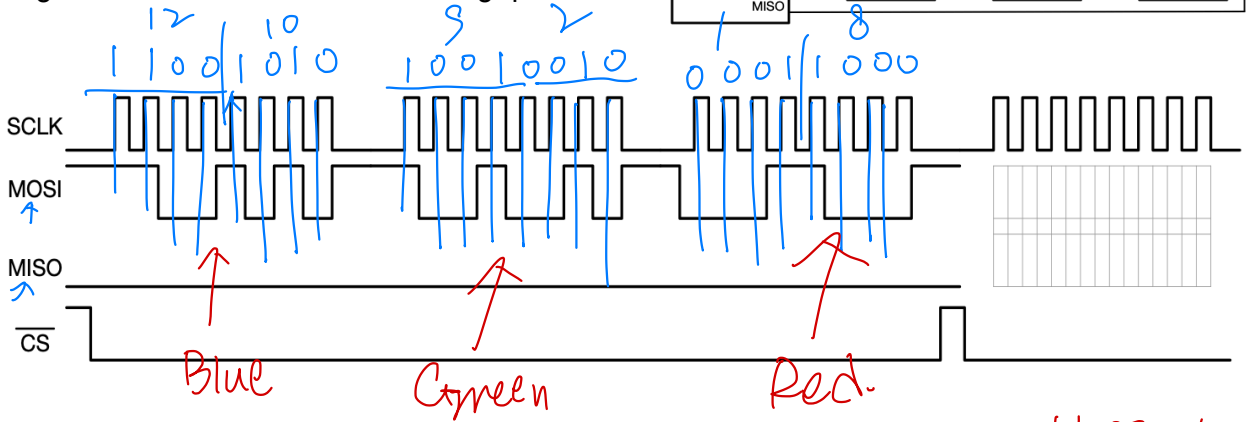
The result in the microcontroller is not setup correctly

packet-size. # bits may be set differently for the sender and receiver. for instance. if the MCU UART is configure to send 1 start bit. 8 bit data. 1 parity bit and 1 stop bit. And the receiver is expecting 1 start bit. 8 data bit. 0 parity bit.

Question 3: SPI Interfaces [10 points]

A microcontroller sets the value of three ultra-high-power LEDs using a SPI interface setup in a ring topology. The LEDs are simple SPI slave devices that control the brightness of the LED using an 8-bit value (LSB first). When the Chip Select line is driven high the LEDs latch the value stored in their internal SPI shift register, into a *Display Register* that controls the LED intensity. It is not possible for the microcontroller to query the state of the *Display Register* as the LEDs are write only. Use the block diagram and timing diagram below to answer the following questions.

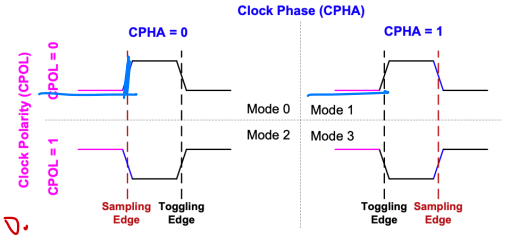




Q3.1 [2 points]

What is the operating mode of the SPI transaction depicted above Fill in the Square.

- [x] Mode 0
- [ ] Mode 1
- [ ] Mode 2
- [ ] Mode 3



Q3.2 [3 points]

When Chip Select line goes high, what are the values (in HEX) that are latched into the LEDs?

Red:

Green:

Blue:

Q3.2 [5 points]

Next, the MCU must update the Red LED to 0xF0 and keep the Green and Blue LEDs at their present values. The fourth frame of the SPI transaction is missing the MOSI and MISO traces. Draw in the missing waveforms. You do not need to draw more than one frame's worth of data.

Question 4: Interrupts [10 points]

Consider a generic ARM Cortex-M4 MCU with the following specifications for interrupt latency:

- 20 cycle latency for entering an interrupt
- Returns from interrupts similarly take 20 cycles
- Tail chaining requires 10 cycles

The following code defines the interrupt priority:

```
4  int main(){
5  /*Main*/
6    //HAL_NVIC_SetPriority(EXTIn_IRQn, pre-emption priority, sub-priority);
7    HAL_NVIC_SetPriority(EXTI1_IRQn, 3, 1);
8    HAL_NVIC_SetPriority(EXTI2_IRQn, 3, 3);
9    HAL_NVIC_SetPriority(EXTI3_IRQn, 3, 2);
10   HAL_NVIC_SetPriority(EXTI4_IRQn, 1, 4);
11   while (1)
12   {
13
14   }
15 }
```

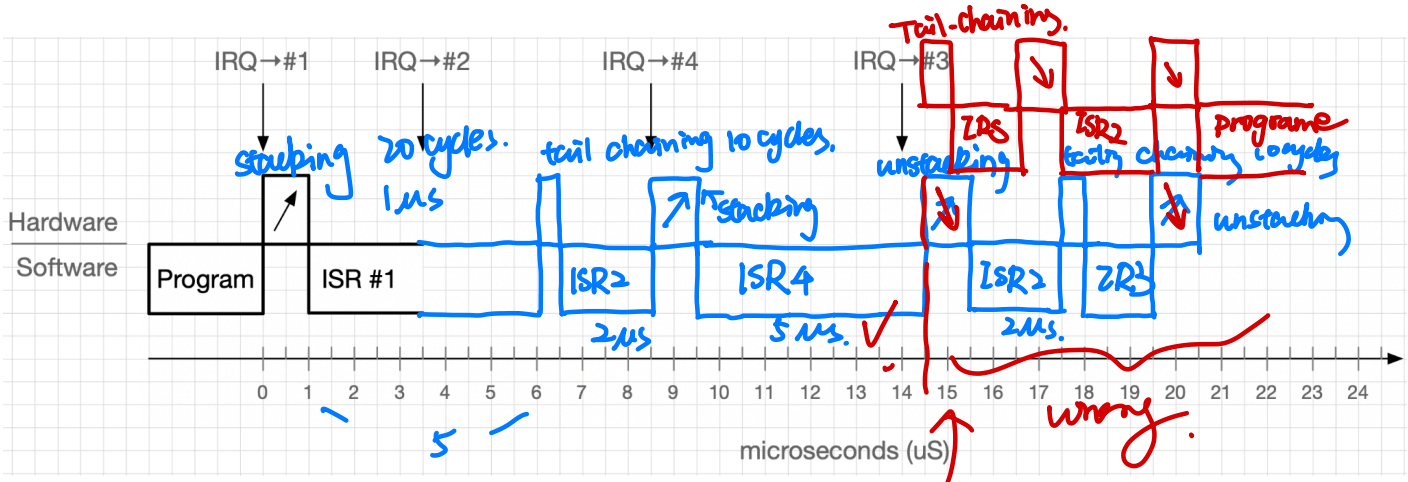
IRQ4 > {IRQ1, IRQ3, IRQ2}

The execution times for the ISRs are:

```
ISR1 = 5us
ISR2 = 4us
ISR3 = 1.5us
ISR4 = 5us
```

Q4.1: [7 Points]

Complete the following timing diagram showing Program and ISR execution. Use ↗ to denote stacking, ↘ to denote unstacking, and ▯ to denote tail changing. Four IRQs are triggered as shown below.



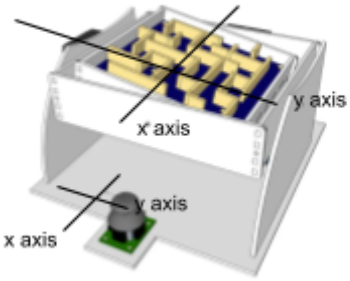
Q4.2: [3 Points]

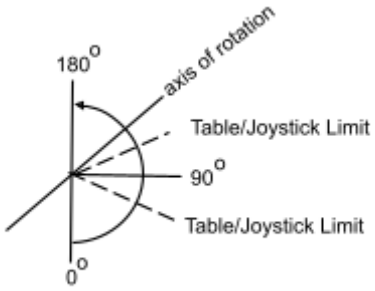
What are late arrivals, and why are they important?

此时有ISR2和ISR3可以被运行. 因为ISR3的sub-priority > ISR2的sub-priority. 所以. 此时不用unstacking回ISR2的值. 而是直接tail chaining到ISR3.

Question 5: Design Problem - Labyrinth Game

This classic game has challenged and fascinated players of all ages ever since it was introduced in 1946. Players turn the knobs to tilt the game board in two axes, with the goal of steering the steel ball from start to finish. The closer you get to the goal, the more points you score. But be careful, or the ball may fall through a hole and send you back to the beginning. You have been tasked with creating a joystick controller system for this game. A typical Labyrinth game has been modified with servos that control each axis. A joystick and microcontroller are used to control the servos, which tilt the platform.





Rotational Coordinate System

System Operation and Coordinate System

The game table top tilts with a one-to-one correspondence joystick position. For example, when the joystick is set with the x-axis rotation to 10 degrees and the y-axis to -5 degrees (10, -5), the tables will be titled by the same angle. The servo motors are attached to the game frame and table table such that when the servo is at 90o the table top is flat, as depicted in Figure 1. The game table and joystick travel are limited to +/- 25 degrees from the level position, as shown in Figure 1.

Components

1. 1x MAT32 processor with associated peripherals.
2. 1x external ADC
3. One Analog Joystick
4. 2x PWM controlled servo motors

Read **All** the Sections Once Before Starting

The following sections consist of the parts you need to complete the design. You will write functions and provide initializations that you will need to implement the final main part of the software control loop. Be sure to read all the sections before starting.

Part 1: GPIO Initialization [4 points]

MAT32F23 has been connected to the two servo motors and the Analog-to-Digital Converter (ADC), as shown in the diagram below. Your task is to Initialize the GPIO peripheral on the MAT32F23 to properly interface with these external devices.

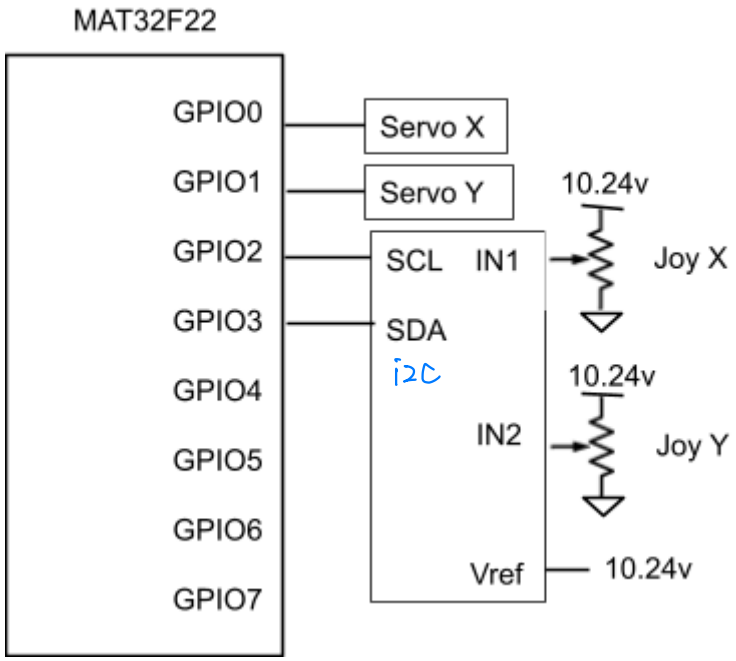


Figure 2

Use the **#defines** provided in the MAT32 GPIO reference to complete the following statement:

```
MAT_GPIO_Init(GPIO_0,

MAT_GPIO_Init(GPIO_1,

MAT_GPIO_Init(GPIO_2,

MAT_GPIO_Init(GPIO_3,
```

Part 2: Joystick [8 points]

Write a function that converts the joystick potentiometer voltage into an angle in degrees. The angle should be in terms of the **joystick's coordinate system**.

The following should be in the datasheet.

Use the following function prototype.

Define a helper function.

float joystick_vtod(float volts)

- The input argument is a floating point number in volts
- The return value is a floating point number in degrees from -25 to +25.

when -25° $V_p = 0$

when 0° $V_p = \frac{1}{2} V_{ref}$

when 25° $V_p = V_{ref}$.

```
float joystick_vtod (float volts).{
    float result = [(volts - 0) / Vref ] x 50 -25
    return result;
}.
```

$\frac{Volts}{V_{ref}} \times 50 - 25.$

$\frac{50}{V_{ref}} \times Volts - 25.$

$\frac{50}{10.24} \times Volts - 25.$

Part 3: Timer Initialization [8 points]

Provide a timer setup function in C that will initialize a timer to provide a PWM signal to a servo with an angular resolution of at least 0.1 degrees and a range of 0 to 180 degrees. Assume the source clock is 10Mhz and the start position is with the table flat.

void position_init(*uint32 timer_base)

- The first argument is an unsigned 32 bit pointer to the timer base register

查看 datasheet：angular resolution：0.1 degree/μs.

period 是 20ms. → $\frac{20}{1000}s$ → $\frac{1000}{20}Hz = 50Hz$.

$20 \times 10^3 \times 0.1 = 2000$ 份

10MHz　　prescalar = 999

$clock = \frac{10MHz}{999+1} = 10kHz$

20 ms = 20,000 μs　　0.1 degree /μs

↓说明.

20,000 counts. at 1μs clock rate.

180° & 0.1°/μs

1800 μs.

Clock 1μs.

1800 ⇗ CCR = 1800.

10MHz → $\frac{1}{10}\mu s$

prescalar = 9

prescaled clock = $\frac{10MHz}{9+1} = 1MHz \rightarrow 1\mu s$

Since period = 20ms.

So ARR = $\frac{20,000}{1} - 1 = 19999$

完成了基本计算.

Step 1：让 CCR 进入 compare mode.

timer_base 是指向 timer. base register 的. 要想修改其它 register 的值. 则须要基于其地址进行加减.

找 control register.

PWM.

400　　1300

0

offset.

*uint32 control_register = timer_base + 0x00

control_register |= 0b0000.

step2：设置 prescale 的值.

*uint32 prescale_register = timer_base + 0x04.

* prescale_register = 9;

step3：设置 ARR 的值

*(timer_base + 0x06) = 19999;.

step4：设置 CCR 的值.

(timer_base + 0x08) = 1300.

Part 4: Set Servo Position [8 points]

Provide a C function that provides a PWM signal pulse width from a servo angle based on your timer settings and the **servo coordinate system.**

void servo(servo_axis, float angle)

- The first argument is an int indicating the servo axis: 0 is x axis and 1 is y axis.
- The second argument is a floating point value of servo position from 0 to 180 degrees.

void servo(servo_axis, float angle){

给 angle ⟶ PWM 值.

$\frac{angle}{180} \times 1800 + 400.$

$10 \times angle + 400.$

因为返回值是 void. 所以要把值写入 register 中

if (servo_axis = 0x40000100){

uint32

*(0x40000100+0x08) = (10×angle +400)

CCR offset

Part 5: ADC Function [10 points]

Provide a C function that uses the ADC to return the value in volts from the selected input channel. The prototype follows.

float ADC(int channel)

- The first argument is an integer specifying the analog input channel.
- The return value is a floating point number that represents the converted value in volts.

total 8 channel.

float ADC(int channel) (

```
*uint32 select_Register = 0x0;
* select_Regist = 0b00000000;
* select_Registe |= channel;
uint32 Data_high_value = * 0x2 & 0b00000011
Data_high_value = Data_high_value << 8;

uint32 Data_low_value = * 0x1 ;
uint32 A_total = Data_high_value + Data_low_value;
float result_voltage = A_total / (10^2-1) x Vref
                     = A_total / 1023 x 10.2V.
return result_voltage.
```

端在不能自接读.

命要通过 I2C 交流.

MAT_I2C_Send (Address ↓ , data ↓ , size ↓)

Slave address + sub address

0x7 0x0.

Char buf[3]

buf[0] = 0.

buf[1] = (uint_8) channel.

Part 6: Write the Control Loop [4 points]

Write the main control loop using the functions you have written from above. Assume any initialization has already occurred and the C functions from above are globally defined. See Figure 1 above for joystick X and Y axis ADC channel assignments and servo X and Y axis assignments.

main(

Part 6: Write the Control Loop [4 points]

Write the main control loop using the functions you have written from above. Assume any initialization has already occurred and the C functions from above are globally defined. See Figure 1 above for joystick X and Y axis ADC channel assignments and servo X and Y axis assignments.

main(